FlashSim: accelerating HEP simulation with an end-to-end Machine Learning framework

Francesco Vaselli[1,3,*], *Andrea* Rizzi[1,2,**], *Filippo* Cattafesta[1,2], and *Gloria* Cicconofri[1,2]
On behalf of the CMS Collaboration

[1]INFN Pisa
[2]University of Pisa
[3]Scuola Normale Superiore

Abstract. We developed a first prototype of an end-to-end machine learning based simulation framework for arbitrary analysis ntuples at the CMS experiment. Such a framework, called FlashSim, was capable of simulating a wide variety of physical objects with good performance on 1d distributions, correlations and desired physical content when compared to the current state-of-the-art simulation. Current methods are based on MC techniques, computationally expensive and requiring a long time to compute. Our prototype was trained to replicate the samples from state-of-the-art methods through the use of the Normalizing Flows algorithm. It showed compatible results with a speedup of several orders of magnitude. This type of approach opens the way to general, analysis agnostic simulation frameworks which may be able to tackle the challenges of the simulation needs for HL-LHC and future collaborations.

1 Introduction

The simulation necessities of HEP collaborations are expected to increase rapidly as we reach new frontiers in energy and luminosity. In CMS [1], the increasing need for full simulation samples is due to various factors, such as the growing amount of experimental data that we need to represent with our simulated samples, or the need for *alternative*, non-statistically limited samples for systematic variations.

These problems are expected to become even more relevant at the HL-LHC when the trigger rate will increase and the time needed for fully simulating an event will grow as a consequence of the higher number of pile-up events.
Current state-of-the-art frameworks for HEP simulation such GEANT4 [2], rely on numerical MC methods, which are very accurate but expensive in time and computational resources.

Fast and accurate simulation frameworks are needed to successfully tackle the future challenges. Currently, CMS uses two frameworks for fast simulation: *FastSim* produces detailed CMS event descriptions, a fast-simulated "AOD" event (about 500kb/event), that can be employed by any analysis, either directly or through the derived MINIAOD [3] and NANOAOD [4] formats. The second tool, *Delphes*, instead uses parametric smearing of generator level information. It produces high-level descriptions of an event without any detail

*e-mail: francesco.vaselli@cern.ch
**e-mail: andrea.rizzi@cern.ch

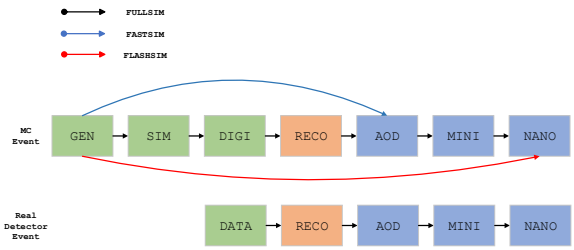

Figure 1: A comparison between the steps of the various simulation frameworks in CMS.

of detector level quantities, i.e., at a similar level of the NANOAOD format.
Novel approaches can take advantage of Machine Learning, which is expected to provide us with both the accuracy and the speed we need. Countless application have already been proposed and explored in the various experiments.

In the present work we introduce a new simulation framework, named *FlashSim*, based on the *Normalizing Flows* ML generative algorithm. It aims to directly produce NANOAOD format samples from generator level information, using generic ML generative techniques rather than a handcrafted, process/analysis specific model. Such an approach sits in between the two existing fast simulation toolkits in terms of detail of the event information, and it is expected to be faster than both of them, especially when accelerated with GPUs. The work presented in this document is a continuation of what was started by the authors in [5]. A similar approach has been since then presented by the ATLAS collaboration in the context of photon simulation in [6]. A more in depth discussion of the present work is also available as a public CMS Note [7].

2 End-to-end simulation with Normalizing Flows

We present the prototype of a general, flexible simulator, not tailored to specific analyses. This is possible thanks to the existence of a common high-level format for analysis (the NANOAOD), that reduces the number of variables to simulate from several thousands to few hundreds; and the recent progress in generative AI, especially in the new algorithm of *Normalizing Flows* (NF) [8].

The idea behind the FlashSim is to go directly from the generator-level content of an event to its NANOAOD representation. This is illustrated in the Figure 1.

One difference compared to typical AI/ML sample generation, e.g. image generation, is that we need to *condition* the generation on some previous information. That is, we do not want to simulate *a generic* CMS event but *the* CMS event corresponding to some given generator level input.
Hence, the natural factorization for the simulation is to go one by one on the various objects, using generator level representation of those objects as conditioning information.

Each object is simulated by a so-called *functional unit*, a transformation implemented by the Normalizing Flows algorithm . At first order, the various units are independent, each taking as input only the relevant physical information for the simulation of its target. However, it may still be necessary to capture correlations that go beyond the per physics object factorization. In order to catch these additional correlations we plan to run the functional units in a chain, so that later units can not only access the generator level information but also the reconstruction information of the previous units.

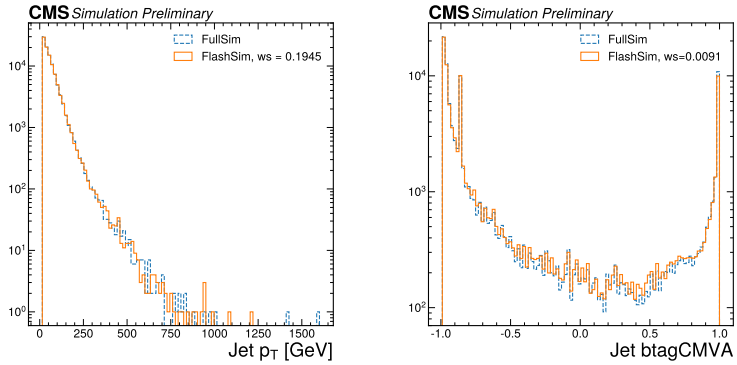

Figure 2: Two out of 17 1D-distributions targeted for the jets. As an additional measure of convergence, we report the *Wasserstein* distance (ws) between the two empirical distributions.

As an example, we can consider the Missing Transverse momentum (MET), i.e. the momentum carried away by a non-interacting particle. A MET functional unit, not yet implemented, would hence need to take as input not only the generator neutrinos but also, for example, the difference in reconstructed versus generated visible momentum, so that the unit itself will only add the effect of unclustered energy.

The final ML models are large, averaging tens of millions of trainable parameters. Despite this, the loss function proved very stable when compared to that of other popular generative algorithms (e.g. GANs). We relied on the Pytorch [9] implementation of Normalizing Flows developed in the package *nflows* [10].

3 Functional units and results

We discuss the various results obtained for each functional unit. The details about the training and the implementation of each functional unit are discussed in detail in a public CMS Note [7].

3.1 Generator matched AK4 Jets

Figure 2 shows the results of this training comparing the distributions of the Jet p_T and the btag *CMVA* score between FullSim and FlashSim. In Figure 3 the same comparison is made for various b/c-tagging algorithms as well as their pair correlations: it is interesting to notice that while FlashSim has no information about the underlying physics that makes some taggers correlated (for example the presence of a secondary vertex), it can correctly reproduce the correlations between different types of taggers.

In order to verify that the model is also learning to properly condition the distribution with the generator information we studied how the btag variables depends on the jet flavour and how the jet response changes as a function of the transverse momentum. Figure 3 compares the efficiency versus mis-tag curve for one of the b-tagging discriminators *DeepCSV* in FullSim, FastSim and FlashSim: FlashSim is much closer to FullSim than FastSim, hence requiring scale factor closer to unity if used in analysis.
Similarly in Figure 4 a comparison of the jet response (bias and standard deviation) is shown for the three types of simulations, again with FlashSim perfectly tracking the momentum

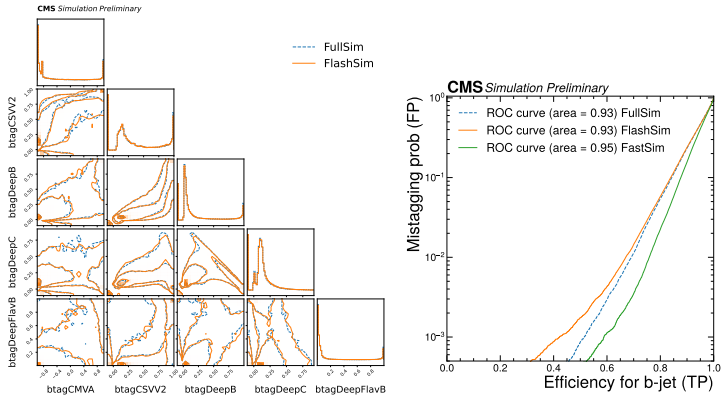

Figure 3: (Left) Comparison of FlashSim versus FullSim for various b- and c-tagging algorithms. On the diagonal di-distributions of the individual variables are shown while the other plots show the correlations among pairs of variables. (Right) Efficiency versus mis-tag (i.e. ROC) for the b-tag algorithm *DeepCSV* comparing fast, full and flash simulation.

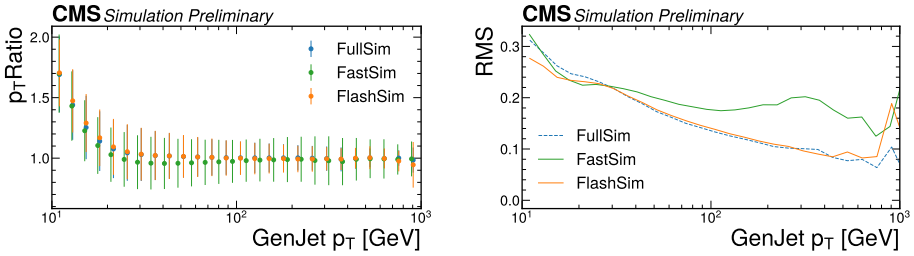

Figure 4: Profile histogram of the jet response (top) and resolution (bottom) as a function of the momentum of the generated jet.

dependency of both the bias and the resolution.

3.2 Fake jets

We call *fake* jets those reconstructed jets that do not have a corresponding generator-level jet. In this case by definition there is no generator jet to start from. As we do not want to simulate PileUp (PU) events generating minimum bias events, we need FlashSim to learn how to produce a variable number of jets with the proper correlations.
In order to generate a variable number of fake jets we use the autoregressive conditioning without permutations, learning the target observables of one fake jet at a time, sorted by p_T. In this way we can train a model that can produce up to N (10 in our tests) jets with a single fixed model. For events with less than N jets the transformations affecting the missing jets are masked so that the relevant weights are not updated. With this architecture, in the simulation phase, the properties of the first jets will be used in the conditioning of the next jets but not vice-versa.
While in training N is fixed by the number of fakes, for the simulation step such number

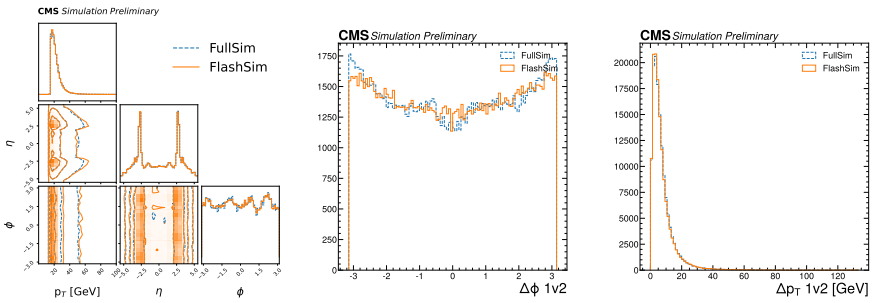

Figure 5: (Leftmost) The target kinematic variables for the leading fake jet of each event show good 1d convergence and capture the correct correlations. (Right) Difference in ϕ (left) and p_T (right) between the first and second fake jets in events with at least two of them.

should be randomly drawn given the PU related variables.

The results of the early prototype are shown in Figure 5 where the kinematic variables and their correlations for the leading fake jet are shown. Figure 5 show the correlation that the network learns between the first and second jets, in particular it should be noticed that the ϕ of the second jet is correctly anticorrelated with the one of the leading jet (given that PU events tend to be balanced in the transverse plane). Similarly the p_T difference plot shows that the model is correctly learning to produce p_T sorted fake jets and is correctly modeling the transverse momentum difference.

3.3 Fat jets

In this case we verified the behaviour of the conditioning looking both at the Particle Net discriminator as a function of the number of b and separating signal (di-Higgs) and background (QCD multijet) events (Figure 6). In addition the softdrop mass is shown separately for signal and background in Figure 6. The small deviations from FullSim can be explained due to the lower statistics available for this training process (order of 100k Phase-2 full-sim events, to be compared with several millions of events available for $t\bar{t}$ Run2 samples used for the other objects).

3.4 Muons

In Figure 7 we show the correlations between the 3D impact parameter and the quadrature sum of the transverse and longitudinal ones: even in this case, where FlashSim has no physics knowledge of these variables (there is no impact parameter of the generated muon passed as input) it correctly models the correlation between these two ways of defining the impact parameter. In the same figure we also verify the goodness of conditioning comparing the impact parameter distributions for different status flags of the generated muons.

3.5 Electrons

The generator level matching of electrons to generator-level particles in NANOAOD is done with no particular selection on the GenPart status, resulting in multiple associations when a

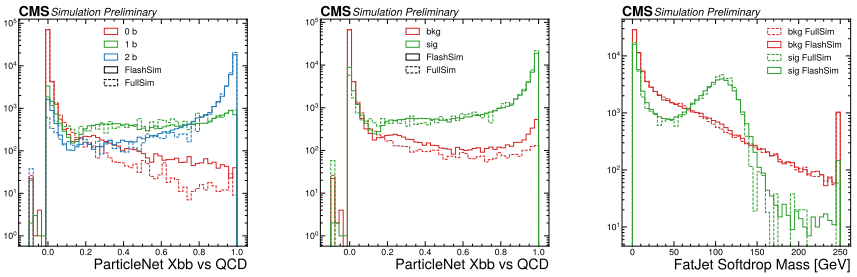

Figure 6: (Left) ParticleNet bb versus QCD discriminator distributions for 0,1 and 2 b-in-jet (left) and for the signal $HH \rightarrow 4b$ versus QCD background (right). The small differences from the FullSim target can be explained as an effect of the low statistics during training. (Rightmost) The softdrop mass is being correctly modeled into its two components for background and signal.

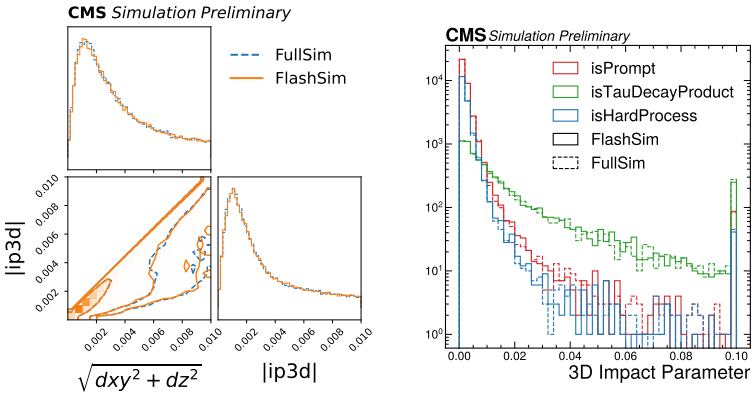

Figure 7: Correlations of different definition of the impact parameters (left) and dependency of the 3D impact parameter from the process generating the muon (right).

GenPart appears in multiple copies. For this reason we implemented a dedicated matching of reconstructed to generated electrons restricting to particles with the *lastCopy* flag set.

Results on 1D distributions and correlations are shown in the corner plot of Figure 8. We can see that FlashSim is obtaining a good closure on FullSim. Even in this case the correlations of shower shape and ID variables are correctly learned despite the complete lack of knowledge of the underlying physics process. The goodness of the conditioning can be instead appreciated in Figure 8 that shows how the distribution of the lepton isolation varies depending on the status flag of the generated electron.

3.6 Benchmark analysis level plots

We generate entire datasets of $t\bar{t}$, *Drell-Yan*, *EWK LLJJ* and *signal* ($H \rightarrow \mu^+\mu^-$) processes, which can be used to repeat the basic steps of a recent analysis to demonstrate the feasibility of our model in a real-case scenario. Events containing both flash-simulated muons and jets are processed with the analysis chain of the Higgs to di-muon VBF analysis (presented

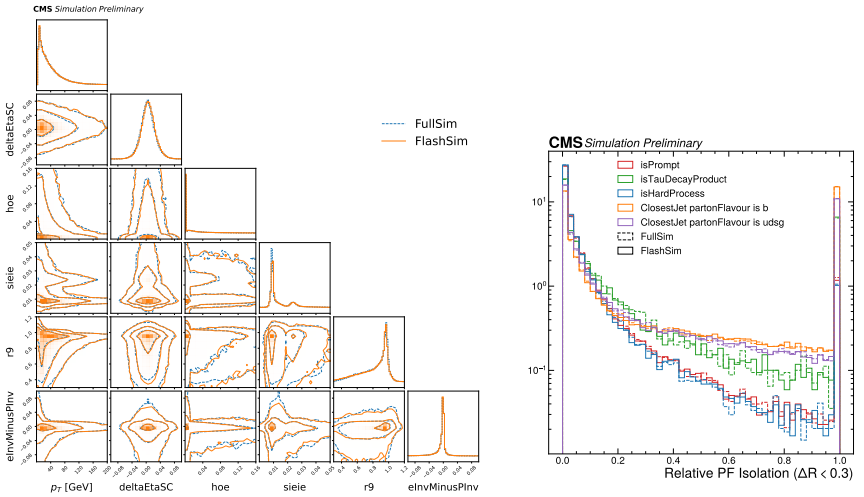

Figure 8: (Left) Comparison of FlashSim versus FullSim for various supercluster variables. On the diagonal di-distributions of the individual variables are shown while the other plots show the correlations among pairs of variables. (Right) The isolation for electrons as a function of the gen electron status flags.

in [11]) in order to verify that the approach we follow produces simulated events usable at analysis level. The DNN used to separate signal from background is calculated starting from flash-simulated quantities and compared between FullSim and FlashSim of the same samples. The DNN uses as inputs multiple derived variables, including some that are correlating the di-jet part of the VBF event with the di-muon part from the Higgs decay. The results are shown in Figure 9.

In addition, FlashSim has been tested to verify that it can be used when a systematic estimate requires evaluating different generators. In Figure 9 (right) we show the effect of evaluating a different signal model with a fully simulated sample or with FlashSim. In the latter case, rather than directly using FlashSim we apply a correction factor *full/flash* obtained from the original signal model. We can see that using this additional correction FlashSim systematic estimate perfectly closes with the FullSim one.

4 Conclusions

We have introduced a prototype for end-to-end simulation of HEP events through novel machine learning techniques. This approach results in several orders of magnitude of speedup (100s/kHz of generation speed per object) when compared to traditional approaches to simulation, while maintaining a good level of accuracy for 1d distributions, correlations and conditioning.

The HEP simulation field is rapidly evolving and moving towards the adoption of ML-based generators across collaborations. However, to the best of our knowledge, this is the first example of a general, analysis-agnostic generator, validated on a wide variety of physical objects and a real analysis benchmark.

The task of tuning and training the various functional units in an unbiased way requires a great effort. Care is needed to ensure roboust modeling of discrete distributions and tails in the continuous distributions. Additionally, multiple trainings would need to be repeated each

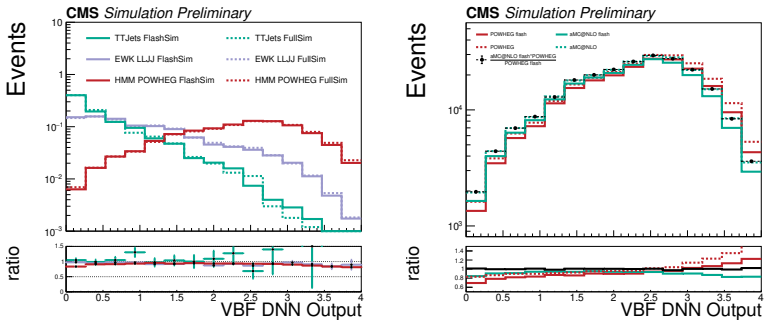

Figure 9: On the left, the comparison of analysis DNN output using both muons and jets information between flash and full simulation on $t\bar{t}$ sample (same process used in training, different events) and signal events (not seen during training). On the right, comparison of systematic shape variations calculated with fully simulated alternative samples and estimate obtained from FlashSim.

time the CMS collaboration releases a new NANOAOD campaign. Finally, this approach is completely dependent on the existence of high-quality, well tuned Full Simulation samples based on GEANT4.

Despite this, no significant obstacles stand in the way of building a complete framework. Such a tool may help greatly when facing the future demands of collaborations in the HL-LHC era, considerably reducing the computational needs and the time taken by simulation tasks.

References

[1] S. Chatrchyan et al. (CMS), JINST **3**, S08004 (2008), also published by CERN Geneva in 2010

[2] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**, 250 (2003)

[3] G. Petrucciani, A. Rizzi, C. Vuosalo, on behalf of the CMS Collaboration, Journal of Physics: Conference Series **664**, 072052 (2015)

[4] A. Rizzi, G. Petrucciani, M. Peruzzi (CMS), EPJ Web Conf. **214**, 06021 (2019)

[5] F. Vaselli, A. Rizzi, *Flashsim: A deep learning solution to the hep simulation problem* (2022), `https://etd.adm.unipi.it/t/etd-08062022-170936`

[6] A. Xu, S. Han, X. Ju, H. Wang, *Generative machine learning for detector response modeling with a conditional normalizing flow* (2023), `2303.10148`

[7] F. Vaselli, A. Rizzi, F. Cattafesta, G. Cicconofri (CMS), Tech. rep., CERN, Geneva (2023), `https://cds.cern.ch/record/2858890`

[8] G. Papamakarios, E. Nalisnick, D.J. Rezende, S. Mohamed, B. Lakshminarayanan, *Normalizing flows for probabilistic modeling and inference* (2021), `1912.02762`

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., *Pytorch: An imperative style, high-performance deep learning library* (2019), `1912.01703`

[10] C. Durkan, A. Bekasov, I. Murray, G. Papamakarios, *nflows: normalizing flows in PyTorch* (2020), https://doi.org/10.5281/zenodo.4296287
[11] A. Sirunyan et al. (CMS), *Evidence for Higgs boson decay to a pair of muons* (2021)